VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2014

Miroslav Kopačka

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Miroslav Kopačka**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: ABB s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. Mgr. Jiří Dvorský, Ph.D.**

Konzultant bakalářské práce: Ing. Oldřich Urbis

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2014

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. 3. 2014

.....

Rád bych na tomto místě poděkoval společnosti ABB s.r.o. za možnost absolvování své bakalařské práxe u nich. Zejména pak Ing. Oldřichu Urbišovi, který mi během praxe dal mnoho užitečných rad a informací, a bez jehož pomoci by tato práce nikdy nemohla vzniknout.
Dále bych rád poděkoval svému vedoucímu bakalařské práce, panu doc. Mgr. Jiřímu Dvorskému, Ph.D., bez jehož pomoci a užitečných rad by tato práce rovněž nikdy nevznikla.

Abstrakt

Tato bakalařská práce obsahuje popis mé činosti ve společnosti ABB s.r.o. V úvodu jsou základní informace o firmě samotné. Dále následuje zadání jednotlivých úkolů a způsob jejich řešení. V závěru zmiňuji vědomosti, které jsem během oněch padesátí dní nabyl, nebo mi chyběli a samozřejmě celkové shrnutí praxe jako takové.

Klíčová slova: Bakalářská praxe, C#, .NET, MVC, Skill matrix, WCF, RoundhousE, PetaPoco, Microsoft, SQL

Abstract

This bachelor thesis describes my work activity in the company ABB s.r.o. In the introduction I will say few information about company in which I have done this bachelor practice. Followed by tasks and their solution. At the end I will summarize skills and knowledges, which I received during the fift;y days in the company followed by completely summary of my action in company.

Keywords: Bachelor practice, C#, .NET, MVC, Skill matrix, WCF, RoundhousE, PetaPoco, Microsoft, SQL

Seznam použitých zkratek a symbolů

HTML	–	Hyper Text Markup Language
CSS	–	Cascading Style Sheets / Kaskádové styly
JS	–	Javascript
ASP	–	Active Server Pages
MVC	–	Model View Controller
Razor	–	View engine pro práci v ASP.NET MVC
WCF	–	Windows Communication Foundation
.NET	–	Software framework vyvinut společností Microsoft
XML	–	Extensible Markup Language
UI	–	User Interface / uživatelské rozhraní
Regexp	–	Regular expression / Regulární výraz
IE	–	Internet Explorer
ORM	–	Object-relational mapping / Objektově relační mapování
SQL	–	Structured Query Language

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Při výběru bakalařské práce jsem měl možnost absolvovat odbornou praxi, místo klasické bakalářské práce. V této možnosti jsem viděl větši přínos pro mou budoucí kariéru a samozřejmě způsob, jak se naučit věcem, které se jinde naučit nedají. Navíc bych tak získal možnost porovnat své dosavadní znalosti ze školního prostředí.

Rozhodl jsem se tedy velice rychle. Vytipoval jsem si hned několik firem, které jsem obeslal e-maily. Některé firmy ani neodpověděly, jiné ano. Jako nejvíce lákavá se jevila firma ABB s.r.o., která si mě následně pozvala na pohovor. K mému velkému potěšení jsem byl přijat a o pár dní později jsem mohl začít pracovat na úkolech, které si pro mě připravili.

2 Odborné zařazení firmy a popis pracovního zařazení

2.1 Odborné zaměření firmy

ABB je mezinárodní společnost, která vznikla v roce 1988 sloučením dvou firem, ASEA (Švédsko) a BBC (Švýcarsko). Tímto spojením vznikla obrovská společnost působící v robotice a zejména v oblastech energetiky a automatizace. Součást ABB je i softwarové oddělení, které krom jiných měst sídlí i v Ostravě. V tomto oddělení pracuje asi 170 zaměstnanců, z toho 30 v Ostravě. Ve světě pak společnost čítá na 150 000 zaměstanců, ve více, než 100 zemích. Podrobnější informace o firmě lze naleznout na stránkách firmy [1].

2.2 Pracovní zařazení studenta

Již při pohovoru se diskutovalo o mém potenciálním pracovním zaměření. Zejména se diskutovalo o platformě .NET a jazyku C#, o mých znalostech v této technologii a samozřejmě mém zájmu rozvíjet se v této oblasti. Diskutovalo se také o problematice, ke které jsem byl po přijetí také přiřazen.

Mým konzultantem se stal Oldřich Urbiš, který mě první den úzce seznámil s danou problematikou. Cílem bylo vytvořit aplikaci, umožnit rychlé a efektivní sdílení informací mezi členy různých týmů. Hlavním prvkem aplikace by byla tzv. Skill matrix, která by umožnila uchovávání, editaci a efektivní vyhledávání v seznamu zaměstnanců a jimi ovládanými technologiemi. V případě problému by pak každý během krátké chvíle věděl, za kým má zajít pro radu, nebo informaci, což umožní zefektivnění práce a zkrácení doby strávené nad řešením problémů.

3 Projekt Knowledge Aggregator

Projekt byl nazván Knowledge Aggregator a hned první den jsem dostal úkoly, na kterých bylo třeba začít pracovat.

3.1 Shrnutí úkolů

3.1.1 Návrh Databáze

Ještě, než jsem začal s prací, bylo nutné pouvažovat nad databází, kterou bych po dobu vývoje používal. Jelikož jsem však věděl, že si databázi můžu bez problému v budoucnu upravovat, mohl jsem se ještě první den pustit na první úkol.

3.1.2 RoundhousE

Jako první úkol jsem dostal zadáno nastudovat si, co to je RoundhousE, k čemu slouží a v poslední řadě aplikovat ho do projektu. Jelikož to pro mě byla úplná novinka, zabral mi tento úkol první dva dny. Nakonec jsem ale zdárně úkol dokončil.

3.1.3 Windows Communication Foundation

Dalším úkolem bylo nastudování služby Windows Communication Foundation (WCF).V případě, že by aplikace byla ve svém konci hodnocena, jako prospěšná, přistoupilo by se k vytváření aplikace na další platformě, v tomto případě na Desktopu (aplikace na PC). K tomuto účelu se WCF skvělé hodí, protože umožňuje používat jednou napsané funkce na více platformách. A to navíc bezpečným a velice jednoduchým způsobem. Tento úkol mi zabral další 4 pracovní dny a to zejména kvůli problémům s referencí na službu a správného pochopení klíčových slov.

3.1.4 ASP.NET MVC

Po úspěšném zprovoznění WCF jsem dostal možnost si vybrat, jakým způsobem budu aplikaci vyvíjet. Na výběr bylo buď ASP.NET MVC nebo ASP.NET Web Forms. S tím, že doporučeno mi bylo využít model MVC. Jelikož jsem v minulosti nikdy nepracoval s MVC modelem a ze školního prostředí jsem měl trochu zkušeností s Web Forms, vybral jsem si ASP.NET MVC. Nicméně zde nastal problém, jelikož jsem neměl potuchy, jak zprovoznit aplikaci vyvíjenou v této technologii. Jelikož jsem neměl moc zkušeností (ne-li žádné) s jazyky HTML, ani JavaScript, které tvoří de fakto základní stavební kámen, trvalo mi asi 7 pracovních dnů, než jsem plně pochopil princip a fungování této technologie.

3.1.5 Vytváření modelů

Jelikož každý projekt založený na MVC by měl obsahovat i svůj model, bylo třeba vyřešit, jak ho udělat v mém případě. Díky službě WCF, která už má stejnou reprezentaci modelů

v sobě, se může zdát, že vytváření stejných modelů ještě jednou a pak do nich překopírovávat data, které už jednou byly načteny, je trochu zbytečné. Nicméně tomu tak není. Úkol mi zabral pouze pár hodin a ještě týž den jsem se pustil do dalšího úkolu.

3.1.6 Přihlašování

Dalším úkolem bylo vyřešit přihlašování uživatelů. Vzhledem k tomu, že jsem řešil například hashování hesel, zabralo mi tato zdánlivě triviální úloha další 3 dny.

3.1.7 Vědomosti

Dalším úkolem bylo umožnit zaměstnancům vkládat, editovat nebo i mazat své dovednosti, na jejichž základě se později bude generovat skill matrix. Tento zdánlivě jednoduchý úkol mi zabral 7 dní neboť jsem neměl žádné vědomosti z Javascriptu, který byl v řešení nepostradatelný a bylo třeba zvolit vhodný způsob zobrazování v tabulkách.

3.1.8 Skill matrix a první dodělávky

Dále jsem potřeboval udělat vhodnou tabulku a view (stránka zobrazovaná uživateli) pro zobrazení skill matrixu a dodělat relativní maličkosti. Následně jsem mohl přemýšlet, zda dalším úkolem bude implementace několikrát zmíněného desktopového klienta nebo potřebný refactoring. Dodělat Webového clienta mi zabralo další 2 dny.

3.1.9 Refactoring

Nakonec jsem dostal jako další úkol provést refactor kódu s tím, že určité funkcionality budu muset opravit. Vyvíjel jsem totiž celou dobu na prohlížeč Google Chrome, nicméně ve firmě je standartem Internet Explorer (IE) 8. Jelikož jsem použil některé funkce z JQuery 2.X, znamenalo to pády aplikace, pokud byla spuštěna v IE. Jelikož jsem u refactoringu musel studovat a použít pár nových věcí, strávil jsem nad nim 14 dní.

3.2 Návrh databáze

Jednalo se o můj první návrh databáze, proto jsem začal tím, že jsem si vyhledal pár informací, co by měla, alespoň v základu, obsahovat. Musel jsem si uvědomit, že aplikace má hlavní myšlenku ve skill matrix, tedy jakási přehledná tabulka zaměstnanců ve firmě a jejich znalostí, a podle toho také musela databáze vypadat. Zaměřil jsem se tedy na to, aby práce se záznamy zaměstnanců a vědomostí byl co nejlehčí. Vznikly tedy tabulky pro ukládání zaměstnanců (*Employee*), vědomostí (*Skill*) a spojovací tabulka m:n vztahu (*EmployeeSkill*). Dále jsem vytvořil tabulky pro ukládání jednotlivých týmu, pozicí členů v týmech a opět spojovací tabulka mezi zaměstnanci a týmy. Poté jsem uznal za vhodné přejít na první úkol a v případě nutnosti databázi upravovat.

3.3 RoundhousE

3.3.1 Co to je RoundhousE

Ve zkratce je RoundhousE nástrojem pro jednoduché verzování databází. Účelem RoundhousE(u) je vyřešit jak problémy spojené s údržbou, tak ulehčit nasazování aplikace do provozu. Využívá obyčejné SQL scripty k přechodu z jedné verze databáze na druhou.

3.3.2 Jak použít RoundhousE

Použití RoundhousE(u) je velice jednoduché. Později ulehčí mnoho práce a použití je opravdu jednocuhé. Všechny důležité informace jsem čerpal z dokumentace RondhousE(u) [8]

V první řadě je dobré si RoundhousE vytvořit v samostatném projektu pro lepší přehlednost, nejlépe typu Library. Dalším krokem je vložení XML souboru, který bude obsahovat minimálně informaci o verzi. V mém případě tento soubor obsahuje krom čísla verze také jméno projektu a jméno společnosti.

```
@echo off
    <?xml version="1.0"?>
–   <buildInfo>
      <projectName>KnowledgeAggregator – Bachelor Thesis</projectName>
      <companyName>ABB CZ</companyName>
      <version>1</version>
    </buildInfo>
```

Výpis 1: Obsah _BuildInfo.xml

Dalším krokem pro správné použití je mít scripty pro DROP a CREATE database a správné hierarchické rozdělení do složek a v neposlední řadě mít k projektu přiřazený rh.exe soubor, který je k dispozici ke stažení z oficiálních stránek RoundhousE(u) [7].

Spouštěcí scripty nutně potřebují pouze jednu informaci a tou je jméno databáze. Pokud se totiž neuvede jméno serveru, databáze se automaticky vytvoří na lokální databázi (localhostu). Spolu s informací o cestě k souboru "rh.exe"můžete vytvořit rychle a jednoduše spouštěcí scripty, jak uvidíte u výpisu kódů 2 a 3.

V první části scriptu je třeba uvést cestu k souboru "rh.exe", za /d se dopíše jméno databáze a za /s jméno serveru. Další informace naleznete v dokumentaci RoundhousE(u) [7].

```
@echo off
echo Ready to drop/create?
pause
".\rh.exe" /f=. /d="KnowledgeAggregator" /s="(localdb)\SQLEXPRESS" /simple /drop /silent
".\rh.exe" /f=. /d="KnowledgeAggregator" /s="(localdb)\SQLEXPRESS" /simple /silent
pause
```

Výpis 2: Spouštěcí script RoundhousE pro DROP/CREATE

```
@echo off
".\rh.exe" /f=. /d="KnowledgeAggregator" /s="(localdb)\SQLEXPRESS" /simple
pause
```

Výpis 3: Spouštěcí script RoundhousE pro UPGRADE

Ke správnému fungování scriptu je potřeba vytvořit složky, s předem stanoveným jménem. Scripty budou tyto složky procházet a hledat scripty, které by měly spustit. Zde uvedu pouze složky, které jsem ve své práci použil, nebo je dobré je zmínit. Další informace naleznete v dokumentaci RondhousE(u) [7].

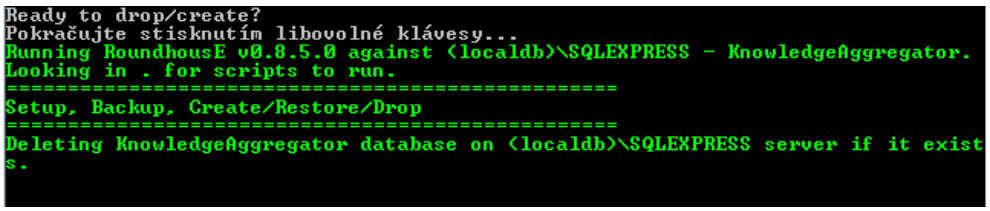
up – Do této složky se dávají scripty, které je potřeba spustit pouze jednou. Pokud se spustí script na drop/create, tak se provedou vždy všechny scripty. Pokud se spustí script pro upgrade, tak se provedou pouze ty, které se ještě nespustili. Pořadí spuštění scriptů se dá dosáhnout abecedním seřazením, např. *0000_Create_aspnet*, *0001_Create_Roles* apod. Je dobré mít na paměti, že RoundhousE si pamatuje, které scripty spustil, proto jakákoliv změna v již spuštěném sql scriptu způsobí, že se bude chtít provést znovu, protože RoundhousE počítá hash souborů, podle čehož vyhodnotí, zda došlo ke změně. To ve většině případů znamená, že se bude snažit vytvořit tabulky, které již existují, nebo vložit data se stejným primárním klíčem.

runFirstAfterUp – Do této složky se dávají scripty, které se spouštějí **vždy** po vytvoření databáze a spuštění scriptů ve složce up.

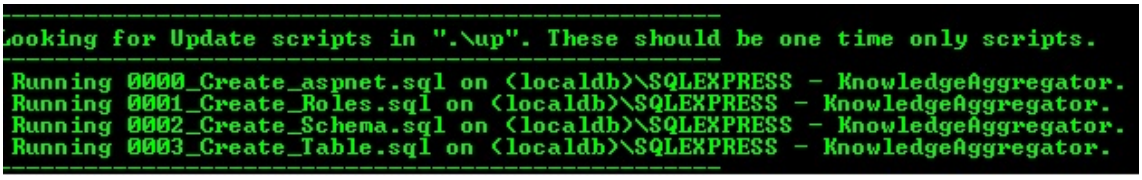
functions – Jak název napovídá, zde se dávají funkce, které budeme chtít v aplikaci využívat.

views – Tato složká slouží pro scripty vytvářející table views.

sprocs – A v neposlední řadě složka pro scripty vytvářející procedury (stored procedures).



Obrázek 1: RoundhousE script - Drop database



Obrázek 2: RoundhousE script - running scripts

3.4 Windows Communication Foundation

WCF je programovací model vyvinutý společností Microsoft pro vývoj servisně orientované (service-oriented) aplikace. Nabízí bezpečné, spolehlivé a jednoduché řešení, jak vyvíjet aplikace na více platforem. Jelikož se jedná o standart společností Microsoft je to také jeden ze způsobů, jak dosáhnout zaručených výsledků. Pro pochopení WCF je dobré projít článek k tomuto určený na C# Corner [3].

3.4.1 Klíčová slova

Zde se pokusím vysvětlit základní označení, které nesmí při tvorbě WCF aplikace v žádném případě chybět. Jedná se o *ServiceContract*, *OperationContract*, *DataContract* a *DataMember*.

ServiceContract se musí označit vytvářený interface. Udává se tím informace, že bude obsahovat Service contract.

```
[ServiceContract]
public interface IDataReader
    {
    ...
```

Výpis 4: WCF - Použití ServiceContract atributu

Dalším v pořadí je **OperationContract**, který udává, že se bude jednat o operaci využívanou službou WCF. Vše, co nebude označeno tímto klíčovým slovem nebude možno nikde použít, protože se nejedná o operaci služby.

```
[OperationContract]
List<UserRepository> GetAllUsers();
```

Výpis 5: WCF - Použití OperationContract atributu

DataContract je jakési ujednání mezi klientskou aplikací a službou, které popisuje data, které mají být vyměněny. To proto, že klient a služba nemusí sdílet stejné datové typy, ale obě strany musí mít stejnou "smlouvu"o posílaných datech. DataContract přesně popisuje parametry nebo návratové typy. Označují se takto třídy, které budou "běhat"mezi klientem a službou.

```
[DataContract]
public class User
    {
    ...
```

Výpis 6: WCF - Použití DataContract atributu

Poslední klíčovým slovem je **DataMember**, kterým se označují property v třídách označenými jako DataContract. Znamená to, že se bude jednat o člena DataContract(u).

```
[DataMember]
public int Id {get; set;}
```

Výpis 7: WCF - Použití DataMember atributu

Když jsem měl toto hotovo, stačilo jen zjístit, jak službu použít v aplikaci. Jak se ukázalo, stačí k tomu jen velice jednoduše přidat referenci na službu (Service reference). Poté si v souboru, resp. třídě, ve které chci službu využívat přidat "using"a v neposlední řadě instanci clienta. Následně jsem byl schopen využívat funkce, napsány ve službě.

Pokud jsem však přidal,nebo upravil nějakou funkci do služby a chtěl jsem využívat pozměněnou verzi, nešlo to. Zde nastal problém totiž v tom, že po každé úpravě uvnitř WCF je nutné provést **rebuild** a poté zvolit **update service reference**. Pokud jeden z kroků bude vynechán, změny se nijak neprojeví mimo službu. Dalším důležitým poznatkem je, že pokud rebuild neproběhne v pořádku, nebo ve WCF projektu budou nějaké chyby, přidat nebo updatovat referenci na službu neproběhne správně, naopak bude vyhozena chybová hláška.

Ještě doporučuju využívat WCFTestClient.exe, který vám dokonale ukáže, zda WCF funguje, nebo ne. Můžete tam napodobovat volání funkcí a porovnávat výsledky. Jak přesně WCFTestClient funguje si můžete přečíst na Microsoft developer network (MSDN) [6].

3.5 ASP.NET MVC (Model View Controller)

3.5.1 Co to je Model View Controller

MVC rozděluje aplikaci do tří hlavních částí – Model, View a Controller. Jako hlavní materiál pro pochopení problematiky se jeví tutorial od firmy Microsoft [5].

Model – objekty modelu jsou část aplikace, která implementuje logiku. Modely často komunikují s databází, ale v mém případě komunikují pouze se služvou WCF, která na základě dat z modelu pracuje s daty uložené v databázi.

View – toto je část aplikace, která zobrazuje uživateli nějaké uživatelské rozhraní (UI).

Controller – toto je část, která reaguje na akce uživatelů, pracuje s modelem a nakonec vybírá View, které se má zobrazit. Rovněž posílá tomuto View data, s kterými při vykreslování může pracovat.

3.5.2 Vytváření modelů

Prvně jsem si myslel, že pokud nevytvořím modely přímo ve Web clientovi, ale nechám je pouze ve službě, kde načtu data a ty pošlu do clienta, kde je už nějak zpracuju, bude

vše správně. Jak se ale později ukázalo, není to tak docela pravda. Problém nastal, jakmile jsem potřeboval nastavit nějakou validační podmínku, ať už nějaký Regexp, povolenou délku textu nebo zda je atribut nutné vyplnit před odesláním formuláře. WCF služba to totiž neumožňuje. Vyřešil jsem to tedy vytvořením modelů uvnitř Web clienta a nastavení validační podmínek zde. Do těchto modelů se následně načítají data, poslány službou.

```
[DataContract]
public class UserRepository
    {
        ...
        [DataMember]
        public string  Username { get; set; }
        ...
    }
```

Výpis 8: Model WCF

```
public class UserModel
    {
        ...
        [Required(ErrorMessage = "Username cannot be empty!")]
        [StringLength(10, MinimumLength = 5, ErrorMessage = "Username has to be 5-10
            characters long!")]
        [RegularExpression(@"^[a-zA-Z0-9]+$", ErrorMessage = "Username can contains only
            letters and numbers!")]
        public string  Username { get; set; }
        ...
    }
```

Výpis 9: Model v MVC

3.5.3 Komunikace s databází

Než jsem se pustil do prvního úkolu, musel jsem si promyslet, jakým způsobem budu přistupovat k databázi. Nakonec jsem využil vědomosti ze školního prostředí a vytvořil jsem pomocnou třídu, která tuto komunikaci obstarávala. Dále jsem si chtěl trochu zpříjemnit práci a tudíž jsem zkusil připojení k databázi napsat do konstruktoru a naopak uzavírání spojení do destruktoru.

```
public class UsersRepository : Database
    {
        public UsersRepository()
            : base()
        {
            this.Connect();
        }

        ~UsersRepository()
        {
            this.Close();
        }
    }
```

Výpis 10: Komunikace s databází

Pravdou však je, že tímto řešením jsem způsobil časté pády aplikace. Když jsem chtěl přistupovat k datům, aplikace spadla, protože "objekt byl nedostupný", nebo naopak se mi uzavřelo spojení a znovu se nenavázalo. Nebyl jsem však schopen zjistit, proč to tak dělá a po debatě s mým konzultantem práce jsem toto řešení odstranil a připojování k databázi i ukončování spojení si raději řeším sám.

3.5.4 Přihlašování a odhlašování

Dalším úkolem, na kterém jsem viděl také možnost dobře se zacvičit do této technologie, neboť se zde již jedná o programování, bylo přihlašování. Bohužel jsem nepočítal s tím, že v případě, že by se to ve firmě začalo používat, byla by využívána Windows authentication, tedy přihlašování na základě přihlášeného uživatele do počítače.

Připravit si potřebné metody ve službě WCF a následně je používat v mém Web clientovi už mnoho času nezabralo. Větší problém nastal, jakmile jsem si uvědomil, že heslo, které uživatel napíše by mělo být nějak šifrováno, proto jsem začal hledat způsob, jak toto zajistit.

Prvním způsobem, který jsem našel a zkusil aplikovat bylo šifrování, které je velice jednoduché k použití a navíc stačí použít knihovnu z .NETu *System.Security.Cryptography*. Zde pouze stačí vytvořit nějaké bajtové pole a poté volat funkce, které jako parametr přijimají jak již zmíněné bajtové pole, tak řetězec určený k zašifrování nebo dešifrování. Nicméně výsledkem byl tak dlouhý řetězec, až to nebylo pěkné a v databázi by každé heslo mělo obrovskou délku.

Dalším řešením, které se mi líbilo bylo vytvoření MD5 hash kalkulátoru. Tento kalkulátor by vytvořil jedinečný hash klíč pro každé heslo a to o délce 32 znaků, což bylo podstatně lepší. Zde je opět podstatné použít knihovnu *System.Security.Cryptography*, protože obsahuje *MD5CryptoServiceProvider*. Při počítání MD5 hashe jsem si musel převést řetězec určený k zašifrování na bajtové pole, z kterého se poté pomocí *MD5CryptoServiceProvider* vypočítá MD5 hash, uložený opět v bajtovém poli. Na konec bylo tedy nutné toto pole převést na řetězec, který se následně uložil do databáze.

```
byte[] textBytes = System.Text.Encoding.UTF8.GetBytes(stringToHash);
MD5CryptoServiceProvider md5 = new MD5CryptoServiceProvider();
byte[] returnValue = md5.ComputeHash(textBytes);

StringBuilder sb = new StringBuilder();
   foreach (byte b in returnValue)
      {
         sb.Append(HashCalculator.HexStringTable[b]);
      }
```

Výpis 11: MD5 hash

Vzhledem k typu přihlašování bylo nutné implementovat také registraci, což sice zabralo pár hodin, ale od přihlašování se to už příliš nelišilo.

Když jsem řešení konzultoval, bylo mi řečeno, že stejně v budoucnu bude třeba aplikovat Windows authentication a navíc toto řešení, i když nebylo špatně neřešilo problém, který jsem tím chtěl eliminovat. Šifrování totiž probíhá na straně serveru, nikoliv na straně klienta. V době, kdy uživatel napíše heslo a bude se chtít přihlásit, bylo by odchytitelné. Nicméně jsme se dohodli, že v budoucnu budu dělat také refactoring, tak bych ho mohl spojit s těmito úpravami. A to i přes to, že základní myšlenka refactoringu je neměnit funkcionalitu, ale pouze zpřehledňovat a upravovat kód. Pustil jsem se tedy do dalšího úkolu.

3.5.5 Vědomosti

Prvním krokem v řešení tohoto úkolu jsem si připravil metody pro CRUD operace - insert, update a delete. Následně jsem se pustil do zjišťování informací o tabulkách. Klasické tabulky vykresleny v HTML jsem tavrhnul hned na začátku. Krom samotného vykreslení zde není nic dalšího řešeno, což jsem považoval za značnou nevýhodu, jelikož jsem věděl, že existují komponenty a plug-in(y), které pomůžou vyřešit spoustu dalších věcí.

Jak jsem zjistil, v ASP.NET MVC je zabudována komponenta, která spoustu z problému řeší - např. stránkování, nebo řazení a třízení. Zkusil jsem tuto komponentu tedy aplikovat do projektu. A i když syntaxe nebyla obtížná, měl jsem problém s nastavováním kaskádových stylů (css) a správným zobrazením sloupců. WebGrid má již předdefinované vlastnosti pro css styly, což mi značně ulehčilo práci. Stačí jen k určité vlastnosti přidat třídu, kterou už poté mohu obsluhovat jako klasickou css třídu.

```
@grid.GetHtml(
   fillEmptyRows: false,
   headerStyle: "table-header",
   rowStyle: "table-row",
    ...
```

Výpis 12: Ukázka nastavení tříd u WebGridu

Dále je třeba specifikovat jak budou vypadat jednotlivé sloupce. Zde je opravdu hodně možností, některé snadné, některé naopak docela obtížné, pokud člověk neví správnou syntaxi. Nejjednoduším způsobem je pojmenovat sloupec stejně, jak máme pojmenovanou

propertu v objektu, který vypisujeme. WebGrid už pak automaticky připraví hlavičku sloupce i hodnoty, které zobrazuje.

```
columns: new[] {
            grid.Column("Name")
        })
```

Výpis 13: Column u WebGridu

Pokud však chceme, aby se hlavička sloupce pojmenovala nějak jinak, je třeba provést přetížení. Pokud se přepíše hodnota v názvu sloupce, sice to bude fungovat, ale nebude fungovat řazení, protože k tomu je nutné pojmenování stejné, jako je properta. Správně je tedy toto:

```
columns: new[] {
            grid.Column("Name", "Jmeno")
        })
```

Výpis 14: Column u WebGridu

```
columns: new[] {
            grid.Column("Name", "Jmeno", format: @<label>@item.Name</label>)
        })
```

Výpis 15: Column u WebGridu

V příkladu je vidět, že k jednotlivým záznámům se přistupuje jako k *itemu @item*. Naopak řešení ukázáno níže bude sice fungovat, ale nebude funkční sortování:

```
columns: new[] {
            grid.Column("Jmeno", format: @<label>@item.Name</label>)
        })
```

Výpis 16: Column u WebGridu

Další problém, s kterým jsem se setkal byl, když jsem potřeboval na každém řádku zobrazit jen část řetězce, nebo jinak zpracovat záznam před jeho zobrazením. Po dlouhém boji jsem však zvítězil a řešení je překvapivě lehké.

```
columns: new[] {
            grid.Column("Name", format: SubItem =>
    {
        string result = SubItem.SubString(5);
        return new HtmlLabel(result);
    }
  })
```

Výpis 17: Column u WebGridu

Když už jsem ale měl WebGrid implementován na několika views, objevil se další problém a to ten, že vyhledávání ve Skill matrixu musí být rychlé a efektivní. Nicméně WebGrid funguje na straně serveru. Když jsem přidal textové pole pro vyhledávání, tak každé vyhledávání se muselo potvrdit a poslat na server, který poté vrátil výsledek. Proto jsem

neskončil s hledáním dalších řešení. Nakonec jsem objevil plug-in pro jQuery *DataTables*, který mi vyhovoval daleko více a navíc se zde lépe pracuje s vyhledáváním. Navíc vše probíhá v jQuery, tedy na straně klienta a výsledek je rychlejší a příjemnější pro uživatele.

Stačí pouze stáhnout požadovaný javascriptový soubor z webu DataTables [4], který následně musí být vložen do projektu. Následně bylo třeba vykresit klasickou tabulku, pomocí klasického HTML, ke které musí být přařazeno *id* a poté se v javascriptu zavolá jedná funkce, která tabulku překonvertuje na DataTable. Tato funkce se volá buď s implicitním nastavením a nebo s možným přetížením. Je možno specifikovat vše, od počtu záznamu na stránce až po povolování/zakazování vyhledávání. DataTable poté zvládne bez problémů vyhledávat na key press - tedy už během psaní uživatele v reálném čase. Jediná nevýhoda je, že to stahuje vždy všechna data, které mu pošlete a ne pouze pro zobrazovanou stránku. Ve velkém počtu záznamů pak může chvíli trvat načítání a přetížení této funkcionality není snadné, neboť soubor přiložený do projektu je velice složitý na zorientování. Na druhou stranu, všechny data tahá i WebGrid, i když přetížení je tam mnohem snadnější. Rozhodl jsem se tedy pro DataTable, což se ukázalo jako správné a pro zatím vyhovující řešení.

```
$('#MojeIdTabulky').dataTable();
```

Výpis 18: Použití DataTables s implicitním nastavením

3.5.6 Skill matrix a první dokončování

Aby skill matrix byla co nejvíce uživatelsky přívětivá, bylo třeba vymyslet nějaký vyhovující algoritmus pro generování tabulky. Problém nastal, když jsem chtěl zobrazit informace z několika tabulek, do klasického View v MVC jde totiž poslat pouze jeden objekt modelu. Existuje sice *ViewBag*, který do sebe uloží cokoliv, včetně jakéhokoliv objektu, nicméně ten jsem neznal. Nakonec jsem tedy vytvořil další třídu složenou z několika jiných. Tu jsem navíc upravil tak, aby jedna instance objektu obsahovala vždy jendoho uživatele a všechny technologie, které ovládá. Ty jsem následně seřadil díky algoritmům naučených ve škole. Tabulku už pak bylo snadné vygenerovat.

Abych ještě více zpříjemnil vyhledávání v této tabulce - předpokládal jsem možnost, že v budoucnosti bude zobrazováno mnoho dat - začal jsem studovat javascriptový soubor umožňující použití DataTables. Po velice dlouhé době se mi však podařilo najít funkce, které se provádí před zobrazením vyfiltrovaného obsahu a v nich jsem udělal malou úpravu na zvýraznění buněk. Uživatel pak vizuálně hned viděl, které buňky v tabulce obsahují text, jenž hledá.

Zbylé úpravy už byly relativně malé a obsahovaly zejména dodělání pár tabulek a upravení css stylů.

3.5.7 Refactoring

V rámci refactoru jsem dostal tip na použití PotaPoco, které pomáhá vytvořit vlastní ORM a ulehčí hodně práci s databází, což se ukázalo jako velice dobrý tip a rád jsem se s tím

seznámil. Dále byla potřeba předělat přihlašování, aby fungovalo na základě Windows authentication.

3.5.7.1 PetaPoco

PetaPoco bylo vyvinuto v roce 2011 skupinou *Topten software* pod licencí Apache. Použití je možné zcela zdarma. Přesné znění licenčních podmínek je možno najít na webu od Apache [2].

Jedná se o software, který je schopný vygenerovat ORM místo programátora. Prvně jsem očekával, že to bude pomalejší, než ORM přesně na míru, nicméně k mému překvapení tomu tak není. Práce s databází probíhá velice rychle a musím říct, že v případě častých změn v databázi se dokonale hodí pro svou jednoduchost. Do projektu se přidá rovněž velice snadně. Stačí ho přidat do solutiony (jakýsi kontejnér obsahující jednotlivé projekty) - je vhodné vytvořit nový projekt typu Library. Poté stačí otevřít Package Manager Console (v nabídce Tools - Library package manager) a zadat příkaz *Install-package PetaPoco*. Tím se vytvoří všechny potřebné soubory, krom jednoho - *App.config*, který si musíte přidat, a do něj napsat *Connection string* pro připojení k databázi. V souboru *Database.tt* (Models/Generated/..) se poté dopíše jméno connection stringu do předem určené kolonky. Poté stačí soubor uložit a je hotovo. ORM je vytvořeno.

V případě změny databáze pak stačí jen otevřít soubor *Database.tt* a zmačknout klávesy *ctrl+s*. Tím se ORM aktualizuje.

3.5.7.2 Windows Authentication

Zde se vlastně ani moc nejedná o refactor, protože se mění funkcionalita, nicméně v mém projektu je tento způsob přihlašování požadován, a tak jsem ho provedl spolu s refactoringem. Tento úkol byl pro mě oříšek, nicméně zdárně dokončen. Windows Authentication funguje tak, že vás do aplikace přihlásí na základě vašeho účtu v počítači. Prvním krokem jsem tedy smazal již mnou vytvořenou registraci i přihlášení. Dále jsem našel, že v počítači existuje soubor, který stačí pouze spustit a poté specifikovat databázi, nad kterou se má příkaz provést. (Konkrétně se jedná o soubor *C:\Windows\Microsoft.NET\Framework64\v4.0.30319\aspnet_regsql* (samozřejmě pokud je Windows adresář jinde než na oddílu C, najdete požadovanou složku tam. Framework složka se pak dělí podle verze systemu). Tímto se vytvoří potřebné tabulky, procedury, views, role a schemata. V aplikaci pak bylo třeba přidat uživatele do tabulky *aspnet_Users* a poté bylo možné, aby se uživatel přihlásil do aplikace pod loginem, kterým je přihlášen v PC.

Další problém nastal, jakmile jsem potřeboval resetovat databázi (drop/create). Je totiž celkem nepraktické, když po každém restartu databáze je potřeba spustit soubor pro vytvoření tabulek pro Windows authentication. Naštěstí existuje způsob, jak pomocí Microsoft SQL management studia vytvořit scripty na základě vytvořených tabulek. Takto jsem si zjistil všechno, co k Windows authentication patří a vložil do příslušných složek v RoundhousE(u). Při každém restartu databáze se mi tak rovnou vytvoří i aspnet tabulky, což je pochopitelně mnohem praktičtější a pohodlnější.

3.5.7.3 Internet explorer 8, CSS styly a drobné úpravy

Dalším velkým problémem byla nefunčnost v prohlížeči Internet Explorer. Aplikace byla totiž vyvíjená i testovaná na prohlížeči Google Chrome. Jak jsem byl upozorněn, jednalo se o závažnou chybu, a tudíž jsem musel aplikaci ještě trochu upravit. Hlavní problém byl v používaní JQuery, neboť Internet Explorer 8 ještě nepodporuje novější verze. Když už jsem musel upravit funkce v Jquery, začal jsem je rovnou rozdělovat do příslušných složek. Měl jsem totiž jeden soubor, který obsahoval všechny funkce a to na každé stránce. Přišlo mi vhodnější, aby každá stránka stahovala jen ty data, které potřebuje k chodu. Navíc jsem spoustu funkcí umazal, protože jsem změnil mírně funkcionalitu. Přidávání, editace atp. jsem řešil pomocí vyskakovacích oken, což je řešeno v JQuery, tuto funkcionalitu jsem zaměnil za nové views, což jak se ukázalo také zpřehlední aplikaci. Dále jsem upravil CSS styly, protože vzhled aplikace se nikomu moc nelíbil. Přidal jsem například menu, které zobrazovalo další nabídky až po najetí myši, což také aplikaci zpřehlednilo. Udělal jsem zarovnání do stran a změnil barvy. Vizuální efekt byl až překvapivý.

4 Projekt Motor Service Portal

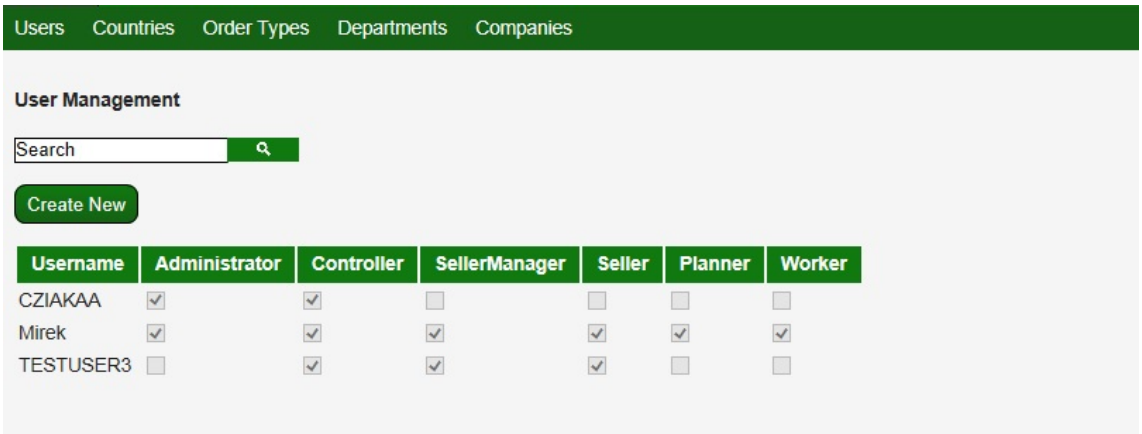
Dále mi bylo oznámeno, že mě zkusí přiřadit na zcela nový projekt, s tím, že osud Knowledge Aggregatoru je nejistý. Motor Service Portal je projekt, který si už objednal reálný zákazník a tudíž se zde jevila i možnost, že v případě spokojenosti bych mohl dostat i nějakou dohodu na budoucí působení ve firmě. Další plus je právě možnost zaučit se a zorientovat v projektu ještě v rámci bakalářské praxe. Zde jsem už strávil všechen zbylý čas určený pro bakalařskou praxi.

4.1 Motor Service Portal

Tento projekt měl za úkol ulehčit správu nad motory, které byly přivezeny do závodu na opravu a to jak z hlediska uchování informací na jednotném místě, tak z hlediska efektivního naplánování potřebných zdrojů a hlavně zpřístupnit práci více lidem na jednou. Do této doby bylo totiž vše uchováváno v Excelech.

Do projektu jsem byl přiřazen, když se měnila struktura databáze, neboť nevyhovala představám o budoucí práci s touto databází. Práce s ní by tedy byla velice obtížná. Databáze se navrhovala pečlivě a to několik dní, protože si každý člen týmu byl vědom, že to v budoucnu ušetří spoustu jiných dní.

Následně jsem mohl uplatnit své vědomosti z Knowledge Aggregatoru, hlavně v oblasti komunikace mezi controllerem a view, neboť aplikace se už vyvíjela v technologii MVC, ale vývojáři přiděleni k tomuto projektu s touto technologii moc zkušeností neměli. Začal tedy vývoj aplikace a prvně jsme se zaměřili na uchovávání informací v databázi, efektivní vkládání nebo editaci.



Obrázek 3: MSP - Začátek administrace

5 Uplatněné, získané a chybějící dovebnosti a znalosti

I když teoretické znalosti získané jak na Vysoké, tak na střední škole byly docela obsáhlé, ukázalo se, že praxe je hodně rozdílná. Přesto si ale myslím, že bez těchto teoretických znalostí bych byl bezradný.

Za nejvíce přínosné teoretické znalosti pokládám znalosti z okruhu databází. Zejména pak připojení a práci s databázi v jazyce C#. Další velice dobrá dovednost, kterou jsem si osvojil díky technickému zaměření škol, je logické myšlení. V mnoha případech bylo třeba strávit mnoho času nad vymýšlením řešení a bez tohoto myšlení bych nebyl schopen vymyslet něco smysluplného. Znalosti z jazyka samotného jsou rovněž nepostradatelnou součástí, bez které bych praxi nikdy nemohl dokončit.

Na druhou stranu jsem si také uvědomil, že mnoho vědomostí mi ještě schází a k dokonalému ovládání klíčových technologií je zapotřebí daleko více, než pár let na škole. Technologie i programovací jazyky se neustále vyvýjí a naučit se vše ve školách je jednoduše nemožné. Nejspíše proto mě dokázaly překvapit problémy s aplikací, které ostatním připadaly jako naprosto jasná věc. Nicméně, jako velký nedostatek, který jsem se ve škole nenaučil byly technologie HTML, CSS, JavaScript a ASP.NET MVC. Musím jít ale do sebe a přiznat, že to je spíše moje chyba, než školní, protože předměty k tomu jsou, pouze jsem si je nezvolil.

6 Závěr

V této práci jsem chtěl věrně vystihnout své působení ve firmě ABB s.r.o v rámci bakalařské praxe. Jsem velice rád, že jsem dostal možnost vykonávat praxi právě v této společnosti. Nebýt této příležitosti, nikdy bych si něuvědomil, jak velké nedostatky mám a že nemám polevovat ve svém usilí zdokonalovat se.

Doufám, že časem se aplikace ještě rozšíří a bude se využívat, ale i kdyby k tomu nedošlo, jsem si jist, že mé vědomosti se zdvojnásobily a přisun informací byl neocenitelný. Jsem také velice rád, že mi firma nabídla další spolupráci, čímž mi umožní další profesní růst.

Kopačka Miroslav

7 Reference

[1] ABB Group *The ABB Group a Automation and Power Technologies*,
Dostupné na: http://new.abb.com/about/abb-in-brief

[2] Apache, Version 2.0 *The Apache Software Foundation,*
Dostupné na: http://www.apache.org/licenses/LICENSE-2.0

[3] C# Corner *Accessing a WCF Service in an ASP.Net MVC Application*,
Dostupné na: http://www.c-sharpcorner.com/UploadFile/krishnasarala/accessing-wcf-service-in-Asp-Net-mvc-application/

[4] DataTables *Jquery plug-in pro tabulky,*
Dostupné na: https://datatables.net/

[5] Microsoft Inc. *Intro to ASP.NET MVC 4*,
Dostupné na: http://www.asp.net/mvc/tutorials/mvc-4/getting-started-with-aspnet-mvc4/intro-to-aspnet-mvc-4

[6] Microsoft Inc. *Microsoft developer network, knihovna MSDN*,
Dostupné na: http://msdn.microsoft.com/default.aspx#fbid=76h5OrpaIiG

[7] RoundhousE *Professional Database Versioning and Change Management,*
Dostupné na: https://code.google.com/p/roundhouse/

[8] RoundhousE *Professional Database Versioning and Change Management - Getting Started,*
Dostupné na: https://github.com/chucknorris/roundhouse/wiki/GettingStarted